

REMARKS

Claims 1-23 were pending. All stand rejected. The applicant has amended claims 1, 9, 12, 14, 15, 16, 17, 20, 21, 22 and 23 and added new claims 24-29. The applicant requests further consideration and re-examination in view of the amendments above and remarks set forth below.

Claims 9, 12 and 17 are amended for consistency with the claim(s) from which they depend. Claims 14, 15 and 16 are amended for consistency with the other dependent claims.

Notice of References Cited:

The Notice of References cited which was included with the office action mailed August 16, 2004 lists the U.S. Patent No. 5,630,128 to Farrell et al. and U.S. Patent No. 5,379,428 to Belo et al. However, U.S. Patent No. 6,567,839 to Borkenhagen et al. is not listed. Therefore, the applicant respectfully request that the Borkenhagen reference be listed on a Notice of References cited.

Objection to the Title:

The examiner objected to the title as not being descriptive. The applicant has amended the title to be sufficiently descriptive of the claims to which the application is directed.

Rejections Under 35 U.S.C. § 112:

The examiner rejected claim 23 as being indefinite for failing to particularly point out and distinctly claim the subject matter that the applicant regards as the invention. Particularly, the examiner stated that the term “the first process” in line 9 lacks antecedent basis.

The applicant has amended claim 23 so that it is no longer indefinite.

Rejections under 35 U.S.C. § 103:

The examiner rejected claims 1-7, 11-16, 18-19 and 22-23 under 35 U.S.C. § 103 as being unpatentable over U.S. Patent No. 5,630,128 (hereinafter “Farrell”).

The applicant has amended claims 1 and 22 to clarify differences between them and the prior art. The present invention is directed toward reconfiguring thread scheduling using a thread scheduler function unit. The thread scheduler functional unit implements reconfigurable scheduling decision functions using parameter values stored in hardware registers. Applicant's specification at para. 37. These values represent the status/progress of the processes running on a plurality of different threads and are changed based on the processing of their associated threads. Applicant's specification at para. 39. In one embodiment, the values are stored in memory mapped registers that receive memory traffic for the processor. Applicant's specification at para. 39. In another embodiment, the values are obtained by a snooping logic that obtains a memory operation snoop from the processor memory bus and maintains a copy of these obtained parameter values in registers. Applicant's specification at para. 41.

Logic functions are performed on the parameter values by the thread scheduler function unit in parallel with and without interrupting the thread processes to determine if thread scheduling should be reconfigured. Applicant's specification at para. 39. An interrupt signal is sent to interrupt the processor if thread scheduling is to be reconfigured. Applicant's specification at para. 38. Clearly, because the processor is not interrupted to determine if the thread scheduling is determined, the processor is only interrupted after determining that the scheduling is to be reconfigured.

The invention overcomes shortcomings of prior thread scheduling techniques. For example, prior techniques that use fixed prioritized interrupts and software based scheduling are unable to dynamically prioritize interrupts in order to decide whether to interrupt a current thread and run the risk of incurring high overhead in evaluating desired scheduling functions. See, applicant's specification at paras. 6-8.

Accordingly, the applicant has amended claim 1 to recite a method for reconfiguring thread scheduling comprising the steps of: obtaining parameter values by monitoring memory operations for a plurality of different threads, the parameter values each modified by execution of a corresponding one or more of the threads by a processor; determining if thread scheduling is to be reconfigured by performing logic functions on the parameter values using a thread scheduler function unit that performs

said determining, in parallel with, but without interrupting execution by the processor of a currently enabled one of the threads and, if so, which one of the threads should be enabled; and sending an interrupt signal to interrupt the processor after determining that thread scheduling is to be reconfigured.

Farrell discloses controlled scheduling of program threads in a multitasking operating system. Farrell, title. Each of the program threads is assigned a priority level and a dispatch class in which the thread resides. Farrell, Abstract. Based on these parameters, the operating system schedules the threads for execution. Farrell Abstract. The operating system selects the highest priority program thread which is available for execution from each dispatch class. Farrell Abstract. An application thread can change the dispatch class in which a program thread resides. Farrell Abstract. Also, an executing program thread can voluntarily yield to a specified thread in the same dispatch class or permit the highest priority available thread in the dispatch class to be queued on the run list with itself. Farrell Abstract.

A ThreadCreate function is called by application programs for each thread to be created. Farrell col. 3, line 52, to col. 5, line 13. The ThreadCreate function positions the threads in each dispatch class according to their relative priority level using thread state descriptors for each thread. Farrell at col. 4, lines 25-49. After a thread is created and organized into a dispatch class, the thread is scheduled by the ThreadCreate function calling a Promote primitive function identifying the dispatch class in which the thread resides. Farrell at col. 5, lines 14-19. The Promote primitive function queues the thread in a run list in priority order. Farrell at col. 5, lines 19-54. The ThreadCreate function then calls the Promote primitive function identifying the dispatch class of the thread which called the ThreadCreate function. Farrell at col. 5, lines 54-56. Thus, for each dispatch class which does not already have a current thread state descriptor identified, the operating system selects the highest priority unblocked and unsuspended thread, denotes the thread as the current thread and queues the thread onto the run list. Farrell at col. 5, lines 58-62.

Because the run list is now changed, it is possible that a thread on the run list has a higher priority than the currently executing thread. Farrell at col. 5, line 66 to col. 6, line 1. Therefore, all the threads in the run list are now permitted to contend for the CPU. Farrell at col. 6, lines 1-2. The ThreadCreate function calls a Switch

primitive function to initiate execution of the highest priority thread on the run list. Farrell at col. 6, lines 2-5. If a thread is currently executing, the state of the thread is saved which will be used to resume execution of the halted thread at a later time. Farrell at col. 6, lines 5-17. Next, the Switch primitive function removes the highest priority thread from the run list and restores the execution state of the thread obtained from the run list to the CPU causing the CPU to resume executing the thread. Farrell at col. 6, lines 18-36. Thus, a thread can be preempted by a higher priority thread at times when the Promote primitive and Switch primitive functions are called. Farrell col. 6, lines 37-41.

Multiple program threads can be executed concurrently on multiple CPUs. Farrell at col. 11, lines 37-38. In this case, whenever the Switch primitive function is called, the Switch primitive function responds by providing the highest priority thread from the run list for execution by the CPU of the calling thread. Farrell col. 12, lines 1-5. As a result, multiple threads from the run list can be executed concurrently by multiple CPUs. Farrell col. 12, lines 5-7.

As amended, claim 1 requires determining if thread scheduling is to be reconfigured by performing logic functions on a parameter values using a thread scheduler function unit that performs said determining in parallel with, but without interrupting execution of a currently enabled one of the threads and sending an interrupt signal to interrupt the processor after determining that thread scheduling is to be reconfigured. In contrast, the CreateThread, Promote primitive and Switch primitive functions of Farrell are executed by the operating system of Farrell. Farrell, Summary. Thus, the same processors 14a-c that execute the threads execute the operating system function calls. As a result, Farrell requires a processor to discontinue processing a thread before executing one of the function calls in order to execute the function call. Therefore, Farrell does not suggest or disclose this feature of claim 1.

For at least this reason, claim 1 is allowable over Farrell. Claims 2-7, 11-16, 18-19 are allowable at least because they are dependent from an allowable base claim 1.

In addition, claim 1, as amended, recites obtaining parameter values by monitoring memory operations for a plurality of different threads, the parameter

values each modified by execution of a corresponding one or more of the threads by a processor. According to claim 1, these are the same parameter values used to determine if thread scheduling is to be reconfigured. However, as explained above the parameters of Farrell that are used for scheduling are the thread priority and dispatch class. Farrell teaches that thread priorities are determined by application programs. Farrell, at col. 5, lines 51-65. Nowhere does Farrell suggest or disclose obtaining such parameter values by monitoring memory operations. Therefore, Farrell does not suggest or disclose this feature of claim 1.

This is another reason why claim 1 is allowable over Farrell. Claims 2-7, 11-16, 18-19 are allowable at least because they are dependent from an allowable base claim 1.

Claim 22, has been amended similarly to claim 1. Thus, claim 22 recites a system for reconfiguring thread scheduling comprising: a first component for obtaining parameter values by monitoring memory operations for a plurality of different threads, the parameter values each modified by execution of a corresponding one or more of the threads by a processor; a second component comprising a thread scheduler function unit for performing logic functions, in parallel with, but without interrupting execution by the processor of a currently enabled one of the threads, on said parameter values to determine if the thread scheduling on the processor should be reconfigured, and if so, which thread should be enabled; and a third component for sending an interrupt signal to interrupt the processor after determining that thread scheduling is to be reconfigured.

As explained above, Farrell, et al. does not suggest or disclose a thread scheduler function unit for performing logic functions, in parallel with, but without interrupting execution by the processor of a currently enabled one of the threads, on said parameter values to determine if the thread scheduling on the processor should be reconfigured sending an interrupt signal to interrupt the processor after determining that thread scheduling is to be reconfigured. This is clear because the CreateThread, Promote primitive and Switch primitive functions of Farrell are executed by the operating system of Farrell. Farrell, Summary. Thus, the same processors 14a-c that execute the threads execute the operating system function calls. As a result, Farrell

requires a processor to discontinue processing a thread before executing one of the function calls in order to execute the function call.

For at least this reason, claim 22 is allowable over Farrell.

As is also explained above, Farrell, et al. does not suggest or disclose obtaining parameter values by monitoring memory operations for a plurality of different threads, the parameter values each modified by execution of a corresponding one or more of the threads by a processor, as recited by claim 22. According to claim 22, these are the same parameter values are used to determine if thread scheduling is to be reconfigured. Nowhere does Farrell suggest or disclose obtaining such parameter values by monitoring memory operations.

This is another reason why claim 22 is allowable over Farrell.

Claim 23 recites performing first logic functions, in parallel with, but without interrupting the processor, on obtained parameter values to determine if thread scheduling on the processor should be reconfigured and which thread should be enabled; and logic for triggering second logic functions to be performed after the first logic functions to determine which thread should be enabled when at least one parameter is updated during a period when the first process is running.

As explained in the applicant's specification in paragraph 59, two types of thread scheduling re-evaluation code may be employed. A first type is used when a change or update to a parameter is detected when the micro engine or microprocessor that performs the scheduling logic functions is not busy. A second type is used when an update to one or more of the parameters is detected when the micro engine or microprocessor is already busy with an earlier re-evaluation function. Farrell does not suggest or disclose such feature. The portions of Farrell cited by the examiner in rejecting claim 23 at col. 6, lines 23-28 and 34-36 simply discuss removal of the highest priority thread from the run list and executing it. Farrell does not suggest or disclose these features of claim 23.

For at least this reason, claim 23 is allowable over Farrell.

The examiner rejected claims 8-10 and 17 under 35 U.S.C. § 103 as being unpatentable over Farrell as applied to claim 1 in view of U.S. Patent No. 6,567,839 (hereinafter "Borkenhagen").

As explained above, claim 1 is allowable over Farrell at least because Farrell does not suggest or disclose obtaining parameter values by monitoring memory operations for a plurality of different threads, the parameter values each modified by execution of a corresponding one or more of the threads by a processor, nor does Farrell suggest or disclose determining if thread scheduling is to be reconfigured by performing logic functions on such parameter values using a thread scheduler function unit that performs said determining in parallel with, but without interrupting execution of a currently enabled one of the threads.

Borkenhagen also does not suggest or disclose these features of claim 1 that are absent from Farrell. For at least this reason, claims 8-10 and 17, being dependent from claim 1, are allowable over Farrell and Borkenhagen, taken singly or in combination.

The examiner rejected claims 20-21 under 35 U.S.C. § 103 as being unpatentable over Farrell in view of U.S. Patent No. 5,379,428 (hereinafter “Belo”).

Claim 20 recites a plurality of memory mapped registers for monitoring memory operations for a plurality of the threads, each of said registers holding a different thread parameter which is modified by execution of a corresponding one or more of the threads. Claim 20 also recites reconfigurable hardware logic connected to receive a plurality of outputs from such registers in parallel and to perform logic functions substantially simultaneously thereon, in parallel with, but without interrupting the processor, to determine if thread scheduling should be reconfigured, and if so, determining which thread should be enabled. As explained above in reference to claims 1 and 22, Farrell does not suggest or disclose these features. Belo does not suggest or disclose these features either. For at least this reason, claim 20 is allowable over Farrell and Belo, taken singly or in combination.

Claim 21 recites hardware snooping logic detecting from memory traffic selected addresses for parameter values for a plurality of the threads, the parameter values each modified by execution of a corresponding one or more of the threads, including a set of registers for holding local copies of said parameter values with said selected addresses, and logic for updating one of said local copies when the address therefor has been detected. Claim 21 also recites reconfigurable hardware logic connected to receive a plurality of outputs from such registers in parallel and to

perform logic functions substantially simultaneously thereon, in parallel with, but without interrupting the processor, to determine if thread scheduling should be reconfigured, and if so, determining which thread should be enabled. As explained above in reference to claims 1 and 22, Farrell does not suggest or disclose these features. Belo does not suggest or disclose these features either. For at least this reason, claim 20 is allowable over Farrell and Belo, taken singly or in combination.

New Claims:


The applicant has added new claims 24-29. New claims 24-26 and 29 recite features of a multiprocessor system, as described in the applicant's specification at paragraph 49. New claim 27 recites that the logic functions of claim 22 are performed by reconfigurable logic, as described in the applicant's specification at paragraph 39. New claim 28 recites that the logic functions of claim 22 are performed by a microengine or microprocessor, as described in the applicant's specification at paragraph 40. Thus, no new matter has been added. New claims 24-29 are allowable at least because each is dependent from an allowable base claim.

Conclusion:

In view of the above, the applicant submits that all of the pending claims are now allowable. Allowance at an early date would be greatly appreciated. Should any outstanding issues remain, the examiner is encouraged to contact the undersigned at (408) 293-9000 so that any such issues can be expeditiously resolved.

Respectfully Submitted,

Dated: Nov. 5, 2004



Derek J. Westberg (Reg. No. 40,872)